Compressed Suffix Arrays and Suffix Trees

Outline

- Reminders
- Motivation
- Compression results
  - Time & Space bounds
- Compressed Suffix Tree
- Compressed Suffix Array
  - Proof of bounds

Reminder - Symbols

- $T = t_1 t_2 \ldots t_{\text{n-1}}$
  - text of length n-1
  - eof symbol # at the nth position
- T[i,n] is suffix i of text T
  - i=1,…,n

Reminder - Symbols

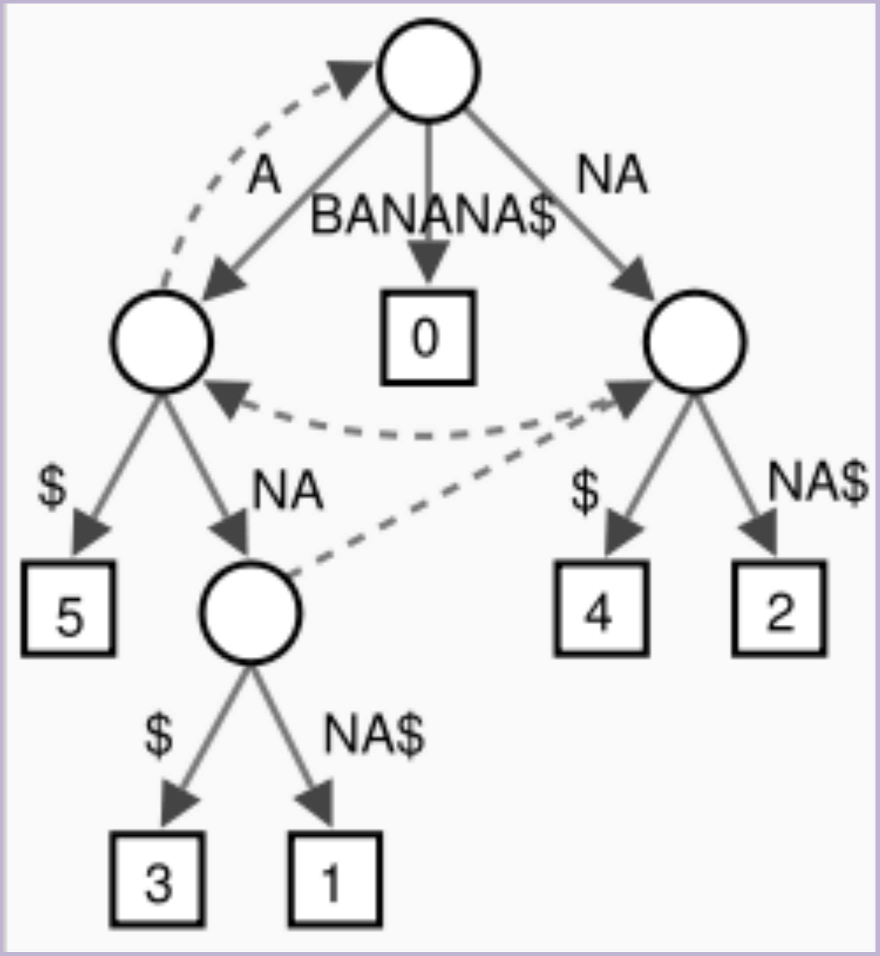
- P = $p_1 p_2 \ldots p_m$
  - pattern of length m
- $0<\varepsilon\leq1$

Reminder - Main Goal

- Search string pattern P within text T
  - Support fast queries
  - Text T being fully scanned only once

Reminder – Suffix Trees

- Leaf with value i represents suffix [i,n]



- Build time
  - O(n)
- Search time
  - O(m)
- Structure space
  - O(n)

Reminder – Suffix Arrays

- Lexicographically ordered
- SA[i] = the starting position in T of the i-th suffix

T = bbba#

1 2 3 4 5

4 5 3 2 1

a#	#	ba#	bba#	bbba#

- Σ={a,b}
- a<#<b

Reminder – Suffix Arrays

- Build time
  - O(nlogn)
- Search time
  - O(m+logn)
- Structure space
  - O(n)

Motivation

- So Far
  - Greedy in space
  - Fast searching

- Need for <u>space-efficient</u> text indexing
- Reduce both space and query time

Compressed Suffix Tree

- Build time
  - $O(n)$
- Search time
  - $O(m/\log n+(\log n)^{\varepsilon})$
- Structure space
  - $(\varepsilon^{-1}+O(1))\, n$

Compressed Suffix Tree

- Build Suffix Array
- Build Compressed Suffix Tree
  - Patricia Tries
- Compress Suffix Array

CSA Basic Operations

- Compress(T,SA)
  - Return succinct representation of SA
  - Retain T
  - Discard SA
- Lookup(i)
  - Return SA[i]
  - Use compressed SA

CSA Primary measures

- Compress
  - Preprocessing compressed SA
  - Space of compressed SA

- lookup
  - Query time

Compressed Suffix Array

- Build time
  - $O(n)$
- Structure space
  - $\frac{1}{2} n \log\log n + O(n)$
- lookup time
  - $O(\log\log n)$

Suffix Arrays Optimization

- Main idea
  - Decomposition scheme
  - Recursive structure of permutations

Decomposition Scheme

- K levels, K=0,….,l
- SA_0 = SA (Original SA)
- n_0=n
  - n = |T|
  - assumption - n is a power of 2
- $n_k=n/2^k$
  - $SA_k=\{1,2,\ldots,n_k)$

SA_k Succinct Representation

- 4 main steps:
  1. Produce bit vector B_k
  2. Map B_k 0’s to 1’s
  3. Compute 1’s for each prefix in B_k
     - Using function $rank_k(j)$
  4. ‘Pack’ SA_k

Step #1: Produce bit vector B_k

- $|B_k| = n_k$
- $B_k[i]=1$ if $SA_k[i]$ is even

  $B_k[i]=0$ if $SA_k[i]$ is odd

T = bba#

SA_0	3	4	2	1
B_o	0	1	1	0

Step #2 : Map B_k 0's to 1's

- New Fuction $\Psi_k(i)$, $i=1,\ldots,n_k$

- $$\Psi_k(i) = \begin{cases} j & SA_k[i] \text{ is odd and } SA_k[j] = SA_k[i]+1 \\ i & \text{otherwise } (SA_k[i] \text{ is even}) \end{cases}$$

T = bba#

SA_0	3	4	2	1
B_o	0	1	1	0
Ψ_o	2	2	3	3

Step #3 : Compute 1's for B_k

- Recall fuction $rank_k(j)$, $j=1,\ldots,l_k$
- $rank_k(j)$ = number of 1's on first j bits of B_k

T = bba#

SA_0	3	4	2	1
B_o	0	1	1	0
$rank_o$	$\boxed{0}$	$\boxed{1}$	$\boxed{2}$	$\boxed{2}$

Step #4 : ‘Pack’ SA_k

- Pack even values of SA_k
  - Divide by 2
- New permutation $\{1,2,..,n_{k+1}\}$
  - $n_{k+1}=n_k/2=n/2^{k+1}$
- Store new permutation into SA_{k+1}
  - Remove SA_k
  - $|SA_{k+1}| = |SA_k|/2$

Example: level 0, steps 1-3

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
T	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b	b	b	a	b	b	a	#
SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30	12	18	27	9	6	3	20	23	29	11	26	8	5	2	22	25
B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	0
$rank_0$	0	1	1	1	1	1	2	3	3	4	4	4	5	6	7	8	9	10	10	10	11	11	12	12	12	12	13	14	14	15	16	16
Ψ_0	2	2	14	15	18	23	7	8	28	10	30	31	13	14	15	16	17	18	7	8	21	10	23	13	16	17	27	28	21	30	31	27

Example: level 0, step 4

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
T	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b	b	b	a	b	b	a	#
SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30	12	18	27	9	6	3	20	23	29	11	26	8	5	2	22	25
B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	0
$rank_0$	0	1	1	1	1	1	2	3	3	4	4	4	5	6	7	8	9	10	10	10	11	11	12	12	12	12	13	14	14	15	16	16
Ψ_0	2	2	14	15	18	23	7	8	28	10	30	31	13	14	15	16	17	18	7	8	21	10	23	13	16	17	27	28	21	30	31	27

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SA_1	8	14	5	2	12	16	7	15	6	9	3	10	13	4	1	11

Lemma : Reconstruct SA_k

- Results of phase k
  - B_k, Ψ_k, $rank_k$, SA_{k+1}
- Reconstruct SA_k

  $SA_k[i] = 2*SA_{k+1}[rank_k(\Psi_k(i))] + (B_k[i]-1)$
  - $i = 1, \ldots .n_k$

Proof, case 1, $B_k[i] = 1$

$SA_k[i] = 2*SA_{k+1}[rank_k(\Psi_k(i))] + (B_k[i]-1)$

- Step #4 : $SA_k[i]/2$ stored in $rank_k(i)^{th}$ entry of SA_{k+1}
  - $SA_k[i] = 2 * SA_{k+1}[rank_k(i)]$
- Step #2 : $\Psi_k(i) = i$

Proof, case 2, $B_k[i] = 0$

$$SA_k[i] = 2 * SA_{k+1}[rank_k(\Psi_k(i))] + (B_k[i]-1)$$

- $\Psi_k(i) = j$
- Step #2 : $SA_k[i] = SA_k[j]-1$
  - $B_k[j] = 1$
- Apply case 1 on j
  - $SA_k[j] = 2 * SA_{k+1}[rank_k(j)]$

Example, case 1, $B_k[i] = 1$

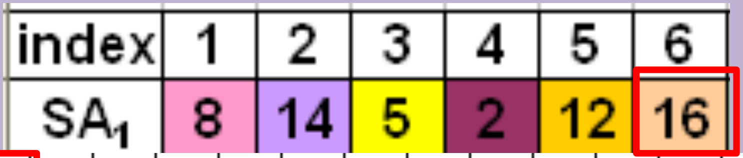
index	1	2	3	4	5
SA_1	8	14	5	2	12

B_0	0	1	0	0	0	0	1	1	0	1	0	0
$rank_0$	0	1	1	1	1	1	2	3	3	4	4	4
Ψ_0	2	2	14	15	18	23	7	8	28	10	30	31

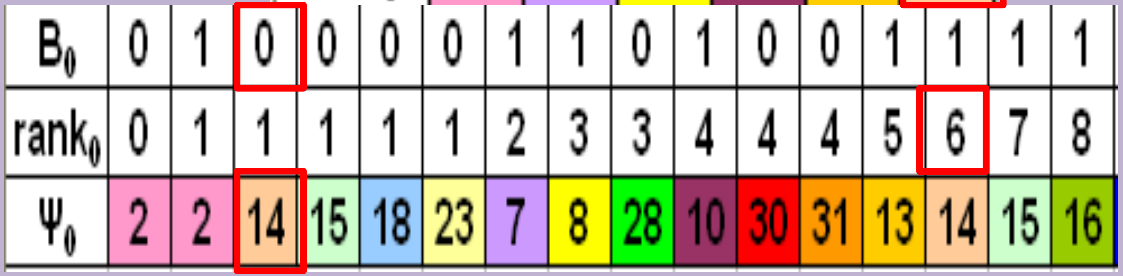
- $SA_0[2]$ = ?
- $B_0[2]=1$, $\Psi_0(2)=2$, $rank_0(2) = 1$
- $SA_0[2]/2$ stored in 1st entry of SA_1
- $SA_0[2] = 2 * SA_1[1] = 2 * 8 = 16$

SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14

Example, case 2, $B_k[i] = 0$

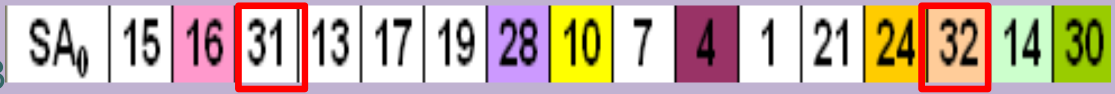


index	1	2	3	4	5	6
SA_1	8	14	5	2	12	16



B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1
$rank_0$	0	1	1	1	1	1	2	3	3	4	4	4	5	6	7	8
Ψ_0	2	2	14	15	18	23	7	8	28	10	30	31	13	14	15	16

- $SA_0[3] = ?$
- $B_0[3]=0$, $\Psi_0(3) = 14$, $rank_0(14) = 6$
- $SA_0[14] = 2 * SA_1[6] = 2 * 16 = 32$
- $SA_0[3] = SA_0[14] - 1 = 32 - 1 = 31$



SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30

Example - Decomposition

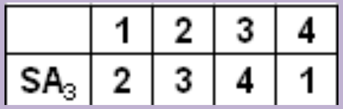
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
T	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b	b	b	a	b	b	a	#
SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30	12	18	27	9	6	3	20	23	29	11	26	8	5	2	22	25
B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	0
rk_0	0	1	1	1	1	1	2	3	3	4	4	4	5	6	7	8	9	10	10	10	11	11	12	12	12	12	13	14	14	15	16	16
Ψ_0	2	2	14	15	18	23	7	8	28	10	30	31	13	14	15	16	17	18	7	8	21	10	23	13	16	17	27	28	21	30	31	27

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SA_1	8	14	5	2	12	16	7	15	6	9	3	10	13	4	1	11
B_1	1	1	0	1	1	1	0	0	1	0	0	1	0	1	0	0
rk_1	1	2	2	3	4	5	5	5	6	6	6	7	7	8	8	8
Ψ_1	1	2	9	4	5	6	1	6	9	12	14	12	2	14	4	5

	1	2	3	4	5	6	7	8
SA_2	4	7	1	6	8	3	5	2
B_2	1	0	0	1	1	0	0	1
rk_2	1	1	1	2	3	3	3	4
Ψ_2	1	5	8	4	5	1	4	8

	1	2	3	4
SA_3	2	3	4	1

Determining l



	1	2	3	4
SA_3	2	3	4	1

  - $n_0 = n = 32$
  - $n_3 = 4 \sim n/\log n$
  - can be stored in ≤ n bits
- Conclusion
  - **$l = \lceil \log\log n \rceil$**

CSA Structure

- **K levels, k = 0,1,….,l-1**
  - Store B_k, Ψ_k, $rank_k$
- Final Level k = l
  - Store only SA_l

CSA Structure & Build

- B_k
  - n_k bits per vector
  - $O(n_k)$ build
- $rank_k$
  - $O(n_k(\log\log n_k)/\log n_k)$ bits
    - As shown before
  - $O(n_k)$ build
- Sa_l
  - $(n/2^l)\log n$ bits

CSA Structure space - Ψ_k

- List method
- 2^K lists
  - possibilities for ‘prefixes’ of suffixes
- Number of lists increases
- L_k = concatenation of all 2^K lists
  - $|L_k| = n_k/2$
  - $|L_k|$ decreases

CSA Structure space - Ψ_k

- For $i = 1,\ldots,n_k/2$
  - $j = i^{th}$ 1 in B_k
  - Pattern in $2^K(SA_k[j]-1),\ldots, 2^K * SA_k[j]-1$ matched to a list

Level 0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
T	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b	b	b	a	b	b	a	#
SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30	12	18	27	9	6	3	20	23	29	11	26	8	5	2	22	25
B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	0
j	0	2	0	0	0	0	7	8	0	10	0	0	13	14	15	16	17	18	0	0	21	0	23	0	0	0	27	28	0	30	31	0

- a list = {2,14,15,18,23,28,30,31}
- b list = {7,8,10,13,16,17,21,27}

Levels 1,2

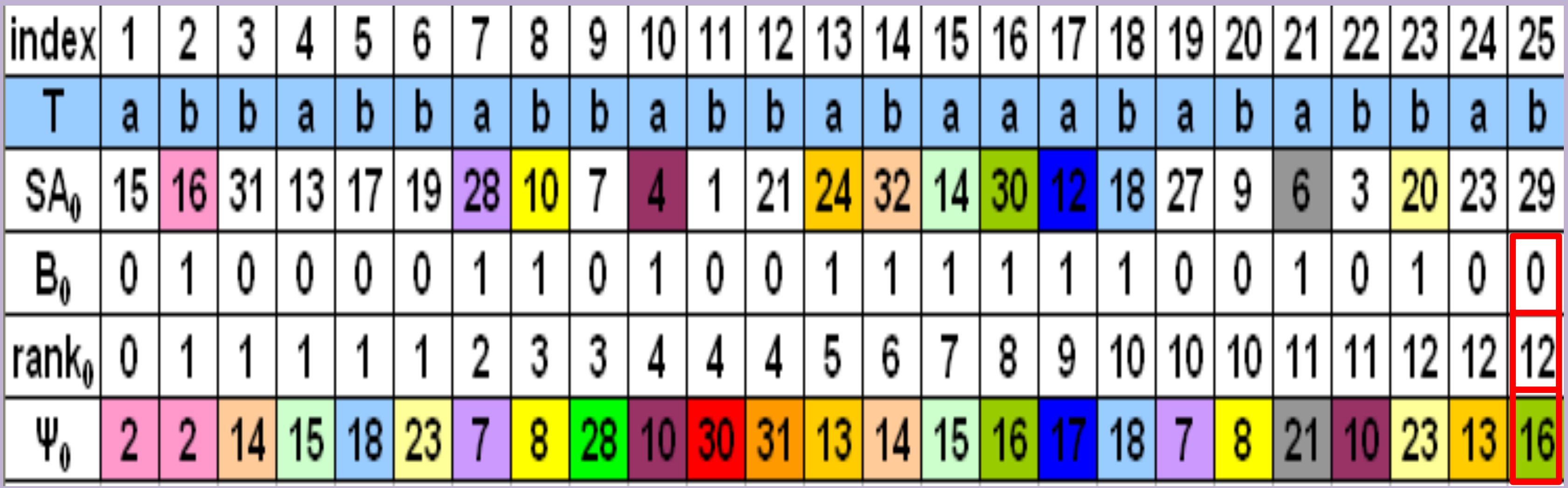
- Level 1
  - aa = {} //empty list
  - ab = {9}
  - ba = {1,6,12,14}
  - bb = {2,4,5}
- Level 2
  - abba = {5,8}
  - baba = {1}
  - aabb = {4}

Reconstruct Ψ_k

- $B_k[i] = 1$
  - $\Psi_k(i) = i$
- $B_k[i] = 0$
  - h = number of 0's in B_k
  - $\Psi_k(i) = L_k[h]$

example : Reconstruct Ψ_k

- $\Psi_0(25) = ?$
  - $B_0[25] = 0$
  - $h = 25 - 12 = 13$
  - $\Psi_0(25) = L_0[13] = 16$



index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
T	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b
SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30	12	18	27	9	6	3	20	23	29
B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0
$rank_0$	0	1	1	1	1	1	2	3	3	4	4	4	5	6	7	8	9	10	10	10	11	11	12	12	12
Ψ_0	2	2	14	15	18	23	7	8	28	10	30	31	13	14	15	16	17	18	7	8	21	10	23	13	16

example : Reconstruct Ψ_k

- $rank_0(16) = 8$
- $SA_1[8] = ?$
- $\Psi_1[8] = ?$
  - $B_1[8] = 0$
  - $h = 8 - 5 = 3$
  - $\Psi_1(8) = L_1[3] = 6$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SA_1	8	14	5	2	12	16	7	15	6	9	3	10	13	4	1	11
B_1	1	1	0	1	1	1	0	0	1	0	0	1	0	1	0	0
rk_1	1	2	2	3	4	5	5	5	6	6	6	7	7	8	8	8
Ψ_1	1	2	9	4	5	6	1	6	9	12	14	12	2	14	4	5

Lemma

- S sorted integers
  - w bits per number
  - $S < 2^w$
- Store integers
  - S(2+w-logs)+O(s/loglogs)
- Retrieve hth integer
  - O(1)

Store L_k

- Store integers
  - $n(1/2+3/2^{K+1})+O(n/2^k \log\log n)$
- Retrieve hth integer
  - $O(1)$
- Preprocess time
  - $O(n/2^k+2^{2^k})$

CSA Structure - Summary

- B_k
  - n_k
- $rank_k$
  - $O(n_k(\text{loglog}n_k)/\text{log}n_k)$
- Sa_l
  - $(n/2^l)\text{log}n$
- Ψ_k
  - $n(½+3/2^{K+1})+O(n/2^k\text{loglog}n)$

Summing it up…

- $\underbrace{nlogn/2^l + \frac{1}{2}l^*n}_{\leq\frac{1}{2}nloglogn+n} + 5n + O(n/loglogn)$

- **<u>½nloglogn + O(n) bits of storage</u>**

Preprocess - summary

- B_k
  - $O(n_k)$
- $rank_k$
  - $O(n_k)$
- Ψ_k
  - $O(n/2^k+2^{2^k})$

- Summing up 0,..,l-1 levels
  - **Preprocess time O(n)**

lookup(i)

- lookup(i) refers to $SA_0[i]$
  - Need to reconstruct $SA_0[i]$
- New procedure - rlookup(i,k)
  - Recursive
  - Based on lemma of reconstructing SA_k

rlookup(i,k)

- rlookup(i,k)

```
If $k = l$
    Return $\mathbf{Sa_l[i]}$
else
    Return $2*\text{rlookup}(\text{rank}_k(\Psi_k(i)),k+1)+(B_k[i]-1)$
```

Reconstruct SA_k

- Lemma
  - $2*SA_{k+1}[rank_k(\Psi_k(i))] + (B_k[i]-1)$
- lookup(i) = rlookup(i,0)

Example - lookup(i)

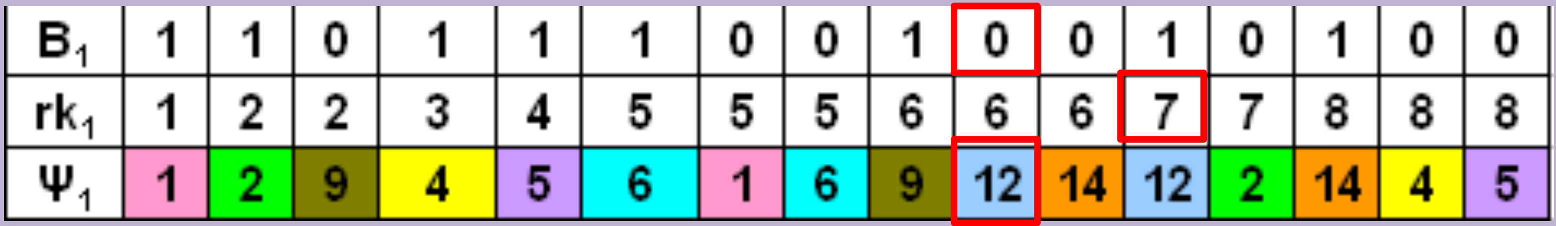
- lookup(5) = rlookup(5,0), l=3
- 2*rlookup($rank_0(\Psi_0(5))$,1)+($B_0[5]$-1)

B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1
rk_0	0	1	1	1	1	1	2	3	3	4	4	4	5	6	7	8	9	10
Ψ_0	2	2	14	15	18	23	7	8	28	10	30	31	13	14	15	16	17	18

- 2*rlookup(10,1)+(-1)

Example – cont.

- rlookup(10,1) = 2*rlookup(rank$_1$(Ψ_1(10)),2)+ (B$_1$[10]-1)

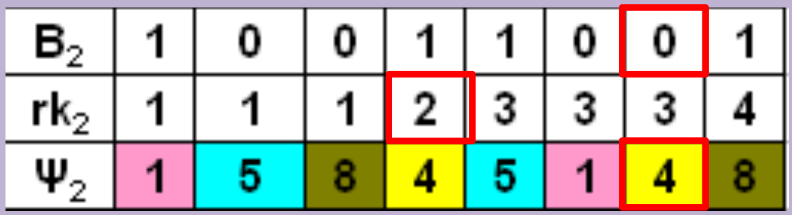


B_1	1	1	0	1	1	1	0	0	1	$\boxed{0}$	0	1	0	1	0	0
rk_1	1	2	2	3	4	5	5	5	6	6	6	$\boxed{7}$	7	8	8	8
Ψ_1	1	2	9	4	5	6	1	6	9	$\boxed{12}$	14	12	2	14	4	5

- 2*rlookup(7,2)+(-1)

Example - cont.

- $rlookup(7,2) = 2*rlookup(rank_2(\Psi_2(7)),3)+(B_2[7]-1)$

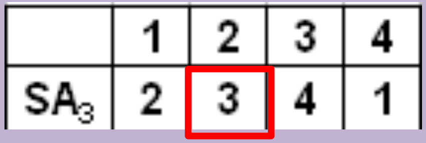


B_2	1	0	0	1	1	0	0	1
rk_2	1	1	1	2	3	3	3	4
Ψ_2	1	5	8	4	5	1	4	8

- $2*rlookup(2,3)+(-1)$

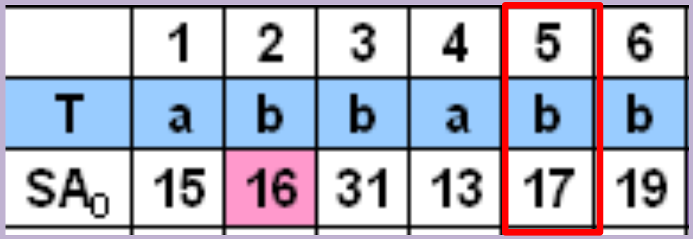
Example - cont.

- rlookup(2,3) =



	1	2	3	4
SA_3	2	3	4	1

- lookup(5) = 2*(2*(2*3+(-1))+(-1))+(-1) = 2*(2*(5)+(-1))+(-1) = 2*(9)+(-1) = 17



	1	2	3	4	5	6
T	a	b	b	a	b	b
SA_0	15	16	31	13	17	19

lookup(i)

- lookup(i) = rlookup(i,0)
  - l+1 levels
  - O(1) per level

  - **<u>O(loglogn) lookup time</u>**

Compressed Suffix Array

- Build time
  - $O(n)$
- Structure space
  - $\frac{1}{2} n \log\log n + O(n)$
- lookup time
  - $O(\log\log n)$

Compressed Suffix Tree

- Build time
  - $O(n)$
- Search time
  - $O(m/\log n + (\log n)^{\varepsilon})$
- Structure space
  - $(\varepsilon^{-1} + O(1))\, n$